How schema mapping can help in data integration ? – integrating the relational databases with ontologies

Maciej Gawinecki
ICT School, Computer Science, XXIII cycle
DII, University of Modena and Reggio Emilia
Via Vignolese 905, 41100 Modena, Italy
maciej.gawinecki@unimore.it

ABSTRACT

Nowadays many application require information from diverse data sources, in which related data may be represented differently. In this paper we briefly present integration of heteregenous data sources and descibe how this process can be supported by schema mapping techniques. We also motivate our theoretical consideration with the practical example of integrating content of a relational database with Semantic Web data.

1. INTRODUCTION

Data integration is the process of combining data residing at different sources and providing the user with a unified view of these data [5]. As a result the user can ask queries in a single unified way over a set of heteregenous data sources.

2. VIRTUAL AND MATERIALIZED DATA INTEGRATION

Generally speaking, integration process can be realized at different levels in the database architecture. This can be observed on the Figure 1 (prepared on the base of [6]). Precisely, the approach, where data coming from local sources are replicated and merged into a new database – a data warehouse – is called *materialized data integration*. In this case queries are executed over unified schema by the data warehouse. Populating the data warehouse is performed during ETL process, i.e. the process of extracting data from local sources, transforming them and loading into the data warehouse. The advantages of this approach are: (a) short response time when quering and (b) avoiding problem of temporal broken connections with local sources. Data integrated into the data warehouse can be also (c) easily filtered, aggregated and annotated. The drawback of this approach is (d) high costs of data storage and maintanance, as the warehouse must be regulary fed with new data and the local data sources must be controlled in respect to the quality of information they provide.

Another approach where data are more loosely coupled, remaining in their local sources, is called *virtual data integration*. In this case we still have a uniform query interface, but a user query is transformed/reformulated into specialized queries over respective original databases (instead of transforming all source data in advance). Obtained partial results are combined and transformed into an answer of expected form. Therefore we can say that data integration takes place "on the fly", when processing query. This approach would be suitable for the situation when (a) there arises a certain global information need and/or (b) we only have a query interface to the data sources and no access to the full data, as it happens e.g. when integrating several commercial query services. This approach has the following disadvantages: (a) process of combining results may require complex operations to eliminate duplicates coming from multiply data sources – online data cleansing; (b) complex query optimisation, i.e. finding a compromise between data completness/quality and response time; and (c) no possibility to update/annotate data in advance.

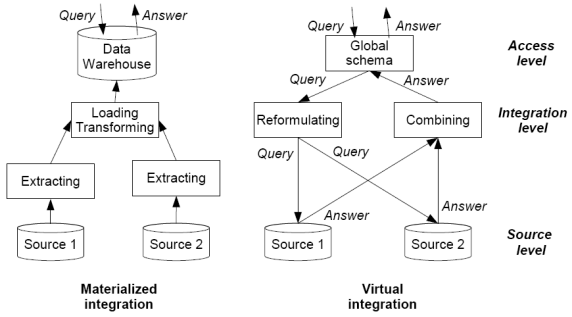

Figure 1: Architecture of systems with materialized and virtual data integration.

3. SCHEMA MAPPING

A *schema mapping* is a set of correspondences between two schemata. Finding a schema mapping is vital for integrating data sources, because it allows to overcome their heterogenity. Particulary:

- data model/representation heterogenity, e.g. heterogenity among relational, XML and RDF models,
- schematic heterogenity, e.g. different data types, different cardinality, normalized vs. denormalized relations, nested structures vs. foreign keys, naming of relations (attributes) etc.

4. SCHEMA MAPPING IN DATA INTEGRATION

In both approaches of data integration uniform query interface is warranted by usage of a single *global schema*. The

problem of its creation can be solved by usage of schema mapping techniques.

4.1 In materialized data integration

In materialized integration approach we usually design a solution in bottom-up fashion. Before we integrate data from various data sources, first we need to integrate their schemata into the global schema. Therefore global schema can be perceived as materialized global view over local schemata (GaV – *Global-as-View*). This step requires us to create mappings between all pairs of the source schemata, and thus determine where they overlap and what are correspondences between concepts they model. Once these mappings are discovered, we can integrate source schemata into the target global schema. Having target schema, we can define mappings between it and particular schemata of original databases. The data warehouse can be populated with source data by interpreting each mapping as a *transformation query*, i.e. a query in a query language (e.g. SQL, SPARQL), which transforms data of the source schema to conform the target schema.

4.2 In virtual data integration

In virtual integration approach design process is determined by an arising information need. Therefore we develop a solution usually in a top-down way, constructing a global schema (hereafter called a *mediated schema*) to best model the kinds of answers the user wants. Next we find mappings between the global schema and respective source schemata. These mappings will be used for the *query unfolding* mechanism, translating a user query into respective subqueries over original data sources. Query unfolding can be realized by a *mediator* – a software component that exploits knowledge about certain local data sources residing under it [9].

In the described method, called also LaV (*Local-as-View*), each local schema can be perceived as a local view over the global schema. Each local data source or group of similar local data sources (e.g. of the same data model) can be wrapped with a *wrapper* component, which simply transforms the local query result into the form conforming the mediated schema. Therefore virtual data integration approach is more flexible then materialized one, as when a new data source is being integrated into the system, the global schema does not need to be re-integrated. Moreover, in materialized data integration approach also data already loaded into the warehouse would need to be transformed to conform the new global schema.

5. APPLYING DATA INTEGRATION

As we observed, schema mapping techniques are tightly bound with data integration process. One of promising domains, where they can be applied is integration of data from legacy databases with Semantic Web ontologies. Data materialized integration approach can be useful for example, when there is a need to use rich expressivity of semantic-based inference over exisiting relational data. On the contrary, virtual data integration can support quering a relational database with use of any ontology query language, e.g. SPARQL. The latter is analised in more detail.

5.1 Inspirating application

In our research we are modelling a set of sensor perceiving moves of people in our department and storing them in a relational database [8]. These databases can be queried with SQL queries. On the other side we have an RDF-triples store, semantically describing features of inmovable, communication devices in different locations of our department. By combining information coming from sensor readings and RDF-triple store we could answer to the questions like: *"What are the devices in the location, where Raffaele was recently seen ?"*. We would like to pose these queries via unified query interface.

Expected answer to a query would be not only identifiers of satisfying devices but also their descriptions, as in the following example (an RDF graph in N3 notation):

```
@prefix vocab:
  <http://www.ing.unimore.it/ecosystem/resource/vocab#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix db: <http://www.ing.unimore.it/ecosystem/resource#> .
@prefix global: <http://www.ing.unimore.it/ecosystem/global#> .
@prefix device: <http://www.ing.unimore.it/ecosystem/device#> .

# knowledge from relational database of sensors readings
[] a global:Observation ;
   global:madeWhen "2008-04-11"^^xsd:dateTime ;
   global:inLocation db:location/1 ;
   global:ofPerson db:person/1 .

db:location/1 a global:Location ;
              global:hasName "Lab202" .

db:person/1 a global:Person ;
            global:hasName "Raffaele" .

# knowledge from RDF-triple store
[] a device:FaxMachine ;
   global:inLocation db:location/1 ;
   device:hasCommunicationNumber "+39 332776501"^^xsd:string .
```

In the presented example we have information not only about `device:FaxMachine`, which is located in the same location – `db:location/1` – where Raffaele was recently seen, but also details of the observation. In fact, this is done by combining of information coming from two different sources. The proposed architecture of data integration is presented on Figure 2 and will be described in detail in subsequent sections. We decided for virtual data integration as we need up-to-date informations from sensor readings.

5.2 Defining mediated schema

On the base of the example answer we can model the mediated schema (hereafter called *the global ontology*), against which we will execute user queries. The schema must be an RDF ontology, which define the following classes: `global:Observation`, `global:Location`, `global:Person`; `global` is a prefix for the namespace reserved for the global schema concepts. We assume that concepts of devices, coming from `device` namespace, are already described in the *device ontology*, i.e. the schema for existing RDF-triple store.

To combine results from the relational database with results from the RDF-triple store we need to define the following mappings: (a) the mapping between the global schema and the schema of relational database and (b) the mapping between the device ontology and the schema of relational database. These are two analogical cases, therefore we will focus only on one of them, the mapping (a).

As the global ontology is defined in RDF data model and sensor readings – in relational data model, first we need to overcome data model heterogenity. Therefore we can define mapping (a) as composition of two mappings:

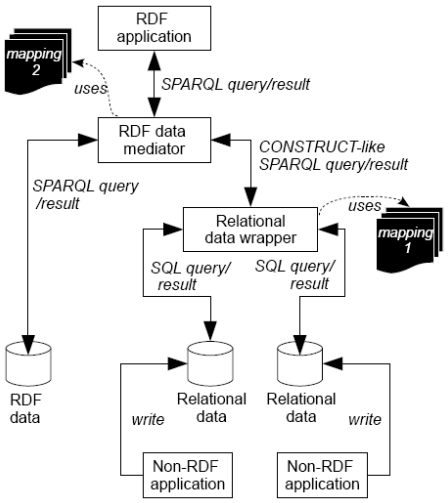

Figure 2: Architecture of our system integrating RDF data and relational data.

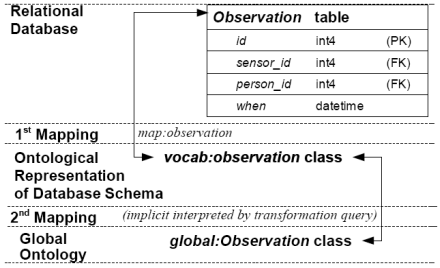

Figure 3: An example of composition of mappings between `observation` table and `global:Observation` class.

- *1st mapping*, which allows to represent relational data in an ontological way;
- *2nd mapping*, which maps ontological representation of the database to the global ontology.

The example of composition of these mappings for the observation table has been given on Figure 3.

5.3 Representing database in ontology

In [2] Berners-Lee proposed very direct mapping: (a) each table is a RDF class, (b) each field (column) name is a RDF property, (c) each record is a RDF node, an instance of the RDF class and so can play a role of a subject in a RDF statement; and (d) each table field (table cell) play a role of a object in such a statement. This mapping should be explicitly defined with a kind of a mapping language. We found two solutions worth of our interest: *OWL.Relational* ontolology [7] and *D2RQ* mapping language [4]. The latter is a part of bigger DR2Q platform [3] for treating non-RDF relational databases as virtual RDF graphs. The platform can generate automatically ontological representation for a given database schema. The following fragment describes the mapping of `observation` and `person` tables.

```
@prefix vocab:
  <http://www.ing.unimore.it/ecosystem/resource/vocab#> .
@prefix map:
  <file:/software/d2r-server-0.4/mapping.n3#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix d2rq:
  <http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#> .

map:observation a d2rq:ClassMap;
  d2rq:dataStorage map:database;
  d2rq:uriPattern "observation/@@observation.id@@";
  d2rq:class vocab:observation .
map:observation_person_id a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:observation;
  d2rq:property vocab:observation_person_id;
  d2rq:refersToClassMap map:person;
  d2rq:join "observation.person_id = person.id" .
map:person a d2rq:ClassMap;
  d2rq:dataStorage map:database;
  d2rq:uriPattern "person/@@person.id@@";
  d2rq:class vocab:person .
map:person_id a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:person;
  d2rq:property vocab:person_id;
  d2rq:column "person.id";
  d2rq:datatype xsd:int .
```

Let us comment the most challenging costraints defined in this mapping:

- *Representing tables as class.* The `map:person` resource introduces a concept of the `vocab:person` class. In D2RQ mapping language introduced classes does not have to represent literally tables from a database, but can have – as properties – columns (and so their values) from different tables. Therefore here we explicitly say that values of the `vocab:person_id` property, attached to instances of the `vocab:person` class, will have values coming from the `id` column of the `person` table.
- *Translation of foreign key relation.* In the database model `person_id` column in `observation` table plays the role of a foreign key referenced to the primary key column `id` in the `person` table. In the ontological representation of the database values of the `vocab:observation_person_id` property will be taken from the `person` table (line: `d2rq:refersToClassMap map:person`) under the condition defined by the `d2rq:join` property.
- *URI identification of resources representing table records.* For example, records of `observation` table will be translated to resources with URIs generated according to the pattern defined by `d2rq:uriPattern` property.
- *Converting datatypes.* Values of the `person.id` column of the `int` SQL datatype will be translated into values of the `int` XML Schema datatype.

The D2RQ query engine, being a part of the platform, can also execute SPARQL queries over the relational database, using the given mapping. In fact, it plays the role of a *relational data wrapper* in architecture of our system (see Figure 2). For example the following SPARQL query:

```
PREFIX vocab:
  <http://www.ing.unimore.it/ecosystem/resource/vocab#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?o ?pn
WHERE {
  ?o a vocab:observation ;
```

```
    vocab:observation_person_id ?p .
  ?p vocab:person_name ?pn .
}
```

should match all observations (instances of the `vocab:observation` class) and names of observed persons (related by `vocab:observation_person_id` property). The query will be translated to the following SQL query:

```
SELECT DISTINCT
  T0_observation.id, T1_observation.id,
  T1_observation.person_id, T2_person.id, T2_person.name
FROM
  person AS T2_person, observation AS T1_observation,
  observation AS T0_observation
WHERE
  (T0_observation.id = T1_observation.id
   AND T1_observation.person_id = T2_person.id)
```

returning a response translated to the following form of a SPARQL query result:

o	*pn*
db:observation/1	Raffaele
db:observation/2	Raffaele

Here we can note two observation about the same person – Raffaele. As it can be seen ontological representation of the database schema is defined in the namespace with the `vocab` prefix and representation of database data – in the namespace with the `db` prefix. The answer is still flat and tabular as it was a result of a SQL query. In the next step we will show, how this can be solved, i.e. how relational data can be easily transformed to a RDF graph.

5.4 Mapping database schema to mediated schema

Using the `CONSTRUCT` clause of the SPARQL query language [1], we can define an answer to the query as a single RDF graph. Therefore this technique can be used to build a transformation query, which transforms the ontological representation of the database data into an instance of the global ontology. The example of such a query, returning part of the expected answer, is presented below.

```
PREFIX vocab:
  <http://www.ing.unimore.it/ecosystem/resource/vocab#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX global: <http://www.ing.unimore.it/ecosystem/global#>

CONSTRUCT {
   [] a global:Observation ;
      global:madeWhen ?w ;
      global:inLocation ?l ;
      global:ofPerson ?r .

   ?l a global:Location ?l ;
        global:hasName ?ln .

   ?r a global:Person ;
      global:hasName "Raffaele"^^xsd:string .
}
WHERE {
  ?r a vocab:person ;
     vocab:person_name "Raffaele" .
  ?o a vocab:observation ;
     vocab:observation_person_id ?r ;
     vocab:observation_sensor_id ?s .
  ?s a vocab:sensor ;
     vocab:sensor_location_id ?l .
  ?l a vocab:location ;
     vocab:location_name ?ln .
}
ORDER desc(?w)
LIMIT 1
```

Here, the `WHERE` clause binds `?r`, `?o`, `?s`, `?l` and `?ln` variables to matching resources in the ontological representation of the database. The `CONSTRUCT` clause uses matched resources to construct the RDF graph describing the last noticed location of Raffaele and details of this observation. The result of this query can be, then, easily combined with knowledge coming from RDF-triple store by the *RDF data mediator* component of the system (see Figure 2).

6. CONCLUSION

In this article we briefly described two approaches of data integration: where they can be used and how they can be supported with schema mapping techniques. The theory has been followed by the practical example of application integrating relational and ontological data together.

7. ACKNOWLEDGMENTS

This paper has been prepared thanks to materials and help provided by prof. Felix Naumann during the "Data Quality and Data Integration" course.

8. REFERENCES

[1] SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/, 2008.

[2] T. Berners-Lee. Relational Databases on the Semantic Web (in Design Issues). http://www.w3.org/DesignIssues/RDB-RDF.html, 1998.

[3] C. Bizer and R. Cyganiak. D2R Server - Publishing Releational Databases on the Semantic Web (position paer). In *5th International Semantic Web Conference*, Athens, USA, 2006.

[4] C. Bizer and A. Seaborne. D2RQ - Treating Non-RDF Databases as Virtual RDF Graphs (position paper). In *3rd International Semantic Web Conference (ISWC2004)*, Hiroshima, Japan, 2004.

[5] M. Lenzerin. Data Integration: A Theoretical Perspective. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART symposium on Principles of Database Systems (PODS '02)*, pages 233–246, New York, NY, USA, 2002. ACM Press.

[6] T. Pankowski. *Bazy danych modele, technologie, narz edzia*, chapter Reformulating queries in data integration systems (in Polish: Reformulowanie zapytań w systemach integracji danych), pages 75–83. Wydawnictwa Komunikacji i L aczności, Warsaw, Poland, 2005.

[7] C. Pérez de Laborda and S. Conrad. Relational.OWL - A Data and Schema Representation Format Based on OWL. In S. Hartmann and M. Stumptner, editors, *Second Asia-Pacific Conference on Conceptual Modelling (APCCM2005)*, volume 43 of *CRPIT*, pages 89–96, Newcastle, Australia, 2005. ACS.

[8] R. Quitadamo, F. Zambonelli, and G. Cabri. The Service Ecosystem: Dynamic Self-Aggregation of Pervasive Communication Services. In *The First Workshop on Software Engineering of Pervasive Computing Applications, Systems and Environments (SEPCASE) at ICSE 2007*, Minneapolis, MN, USA, 2007. IEEE Computer Society Press.

[9] G. Wiederhold. *Readings in agents*, chapter Mediators in the architecture of future information systems, pages 185–196. Morgan Kaufmann Publishers Inc., an Francisco, CA, US, 1997.